Specialization Constants

In Vulkan, all shaders get halfway-compiled by SPIR-V and then the rest-of-the-way compiled by the Vulkan driver. Normally, the half-way compile fixes all constant values and compiles the code that uses them. But, it would be nice every so often to have your Vulkan program sneak into the halfway-compiled binary and manipulate some constants at runtime. This is what Specialization Constants are for. A Specialization Constant is a way of injecting an integer, Boolean, float, or double constant into an already-halfway-compiled version of a shader right before the rest-of-the-way compilation.

That final compilation happens when you call `vkCreateComputePipelines()`. Without Specialization Constants, you would have to commit to a final value before the SPIR-V compile was done, which could have been a long time ago.

Why Do We Need Specialization Constants?

Specialization Constants could be used for:
- Setting the work-items per work-group in a compute shader
- Setting a Boolean flag and then eliminating the if-test that used it
- Setting an integer constant and then eliminating the switch-statement that looked for it
- Making a decision to unroll a for-loop because the number of passes through it are small enough
- Collapsing arithmetic expressions into a single value
- Collapsing trivial simplifications, such as adding by zero or multiplying by 1

Specialization Constants are Described in the Compute Pipeline

In the compute shader

```cpp
VkPipelineShaderStageCreateInfo

layout( constant_id = 7 ) const int ASIZE = 32;
```

In the Vulkan C/C++ program:

```cpp
int asize = 64;
```

```cpp
// one array element for each Specialization Constant
// array this one item is
// size of just this Specialization Constant

// array all the Specialization Constants together
```
In the C/C++ program:

```c
int numXworkItems = 64;

VkSpecializationMapEntry vsme[1];
vsme[0].constantID = 12;
vsme[0].offset = 0;
vsme[0].size = sizeof(int);

VkSpecializationInfo vsi;
vsii.mapEntryCount = 1;
vsii.pMapEntries = &vsme[0];
vsii.dataSize = sizeof(int);
vsii.pData = &numXworkItems;
```

In the compute shader:

```c
layout(local_size_x = 12) in;
layout(local_size_x = 32, local_size_y = 1, local_size_z = 1) in;
```

In the C/C++ program:

```c
int numXworkItems = 64;

VkSpecializationMapEntry vsme[1];
vsme[0].constantID = 12;
vsme[0].offset = 0;
vsme[0].size = sizeof(int);

VkSpecializationInfo vsi;
vsii.mapEntryCount = 1;
vsii.pMapEntries = &vsme[0];
vsii.dataSize = sizeof(int);
vsii.pData = &numXworkItems;
```